J Comput Virol
DOI 10.1007/s11416-006-0009-x

ORIGINAL PAPER

Malware pattern scanning schemes secure against black-box analysis

Eric Filiol

Received: 9 December 2005 / Accepted: 25 March 2006

Abstract As a general rule, copycats produce most of malware variants from an original malware strain. For this purpose, they widely perform black-box analyses of commercial scanners aiming at extracting malware detection patterns. In this paper, we first study the malware detection pattern extraction problem from a complexity point of view and provide the results of a wide-scale study of commercial scanners' black-box analysis. These results clearly show that most of the tested commercial products fail to thwart black-box analysis. Such weaknesses therefore urge copycats to produce even more malware variants. Then, we present a new model of malware detection pattern based on Boolean functions and identify some properties that a reliable detection pattern should have. Lastly, we describe a combinatorial, probabilistic malware pattern scanning scheme that, on the one hand, highly limits black-box analysis and on the other hand can only be bypassed in the case where there is collusion between a number of copycats. This scheme can incidentally provide some useful technical information to malware crime investigators, thus allowing a faster identification of copycats.

1 Introduction

Malware detection in most commercial scanners highly relies on string pattern search. In contrast to the claims usually put forward by most antivirus software publishers, practical experience shows that pattern-based detection techniques are widely used:

- Either as direct detection techniques (first generation techniques like string scanning, wildcards, mismatches...). As an illustration, this kind of techniques is widely used by *On-demand scanners*;
- Or they take place as a final, decisive step once other detection techniques have been first applied, both in On-demand scanners or *On-access scanners* (second generation scanning techniques, malware specific detection techniques, static decryptor detection, code emulation...).

Because of this above–mentionned strong dependence on malware pattern detection techniques – scanning for strings – every commercial antivirus software can be bypassed once a given malware pattern has been extracted and identified through a simple black-box analysis. This problem has previously been initiated [2] even though very simple instances of this problem have been taken into consideration.

A great deal of malware samples that are currently spreading nowadays are produced by copycats from an original malware strain or from its already detected variants.[1] Let us take a significant example: about 27 variants of *W32/Bagle* have been produced from the original code. This proliferation of malware variants not only makes life far more difficult from the computer users' point of view and affects the general security of

E. Filiol
Ecole Supérieure et d'Application des Transmissions
Laboratoire de virologie et de cryptologie
B.P. 18, 35998 Rennes Armées, France
e-mail: eric.filiol@esat.terre.defense.gouv.fr
e-mail: Eric.Filiol@inria.fr

[1] Virus prevalence table regularly illustrates this fact. As an example, 2005 and 2006 statistics [9] show that more than 60% of most prevalent malware were variants of known codes such as *W32/Bagle*, *W32/Netsky* or *W32/Mytob*.

networks but also significantly increases the malware analysts' amount of work in AV companies. These companies who have too much work as it is, do not stop updating their malware database[2] and share their technical information with other AV companies. As a result, the general computer and network protection are far from being acceptable, as is pointed out by the British Department of Trade and Industry in its 2004 Information Security Breaches Survey [22].

More surprisingly, AV companies never made any attempt nor effort to thwart black-box analysis and copycats' activities. Somehow, this attitude encourage copycats to produce malware variants. In this respect, AV companies are partly responsible for malware proliferation. It is all the more surprising that official, reference technical recommendations for protection profile [24] have not considered resistance against black-box analysis as a target of evaluation (TOE).

In this paper, we study the general problem of malware pattern extraction. For that purpose, we have first defined a new model for pattern detection based on Boolean functions under their disjunctive normal form (DNF). Moreover, we have identified some properties that efficient malware detection patterns should at least share, such as resistance against pattern extraction. Our model enables black-box analysis and malware detection pattern to be considered as a DNF learning problem. While the issue of determining whether DNF formulae are learnable or not is still an open problem, we show that available commercial scanners represent very weak instances of this problem. The extraction of detection patterns is actually very easy perform.

We then present a new malware pattern scanning scheme that offers a good resistance level against black-box analysis under the assumption that a significantly large number N of copycats do not collude. We give some constructions in which the parameter N can be chosen according to the desirable level of resistance. Lastly, we provide possible implementations of this scheme which enable computer crime investigators to trace copycats that have been involved in the production and spreading of malware variants.

The paper is organised as follows. In Sect. 2, we first present our mathematical model for malware detection pattern and then detail the properties that reliable detection patterns should inevitably exhibit. Section 3 is devoted to the malware pattern detection problem and presents some mathematical measures of the existing products. Section 4 deals with the detection pattern scanning scheme secure against black-box extraction. Finally, Section 5 contains both a general conclusion and a summary of different open problems which have been identified in relation to the malware detection problem.

2 Mathematical model for malware detection pattern

In this section, our purpose is to define what a *malware pattern* (or *signature*[3]) is. Our definition slightly differs from generally accepted ones. The purpose is to consider at the same time both defence (efficient detection) and attack (copycats' action). Another advantage of our general definition is that it enables to study which properties should exhibit reliable malware detection patterns.

2.1 Mathematical definition of malware patterns

Let us consider a file $\mathcal{F}$ of size n. It describes a potentially infected file or a malware sample. $\mathcal{F}$ is then a sequence of bytes, in other words, a sequence of n symbols over $\Sigma = \mathbb{N}_{255} = \{0, 1, \ldots, 255\}$. Hence $\mathcal{F}$ is an element of Σ^n.

We describe $\mathcal{S}_{\mathcal{M}}$ a malware pattern of size s with respect to a given malware $\mathcal{M}$ as a finite sequence of Σ^s. Thus

$$\mathcal{S}_{\mathcal{M}} = \{b_1, b_2, \ldots, b_s\}.$$

The ith byte of $\mathcal{F}$ (respectively of $\mathcal{S}_{\mathcal{M}}$) will be denoted $\mathcal{F}(i)$ (resp. $\mathcal{S}_{\mathcal{M}}(i)$).

We say that $\mathcal{F}$ is infected by malware $\mathcal{M}$ if they are s locations in $\mathcal{F}$ (byte indices), denoted $\{i_1, i_2, \ldots, i_s\}$ such that

$$\mathcal{F}(i_j) = b_{\sigma(j)} \quad 1 \leq j \leq s,$$

where σ denotes a bijective permutation over the byte of $\mathcal{S}_{\mathcal{M}}$. The use of permutation σ enables taking into account potential modifications of $\mathcal{M}$ made by any copycat. Indeed, different code obfuscation or polymorphism/metamorphism techniques [7,23] can modify the structure (byte ordering or indexing) of $\mathcal{S}_{\mathcal{M}}$:

- By modifying $\mathcal{S}_{\mathcal{M}}$ byte indexing (e.g. dummy code insertion). Then, $\sigma = Id_{\mathbb{N}_s^*}$;
- By modifying $\mathcal{S}_{\mathcal{M}}$ byte ordering (e.g. by obfuscation). Then $\sigma \neq Id_{\mathbb{N}_s^*}$. However the execution flow re-orders bytes of $\mathcal{S}_{\mathcal{M}}$ during the execution of $\mathcal{F}$.

[2] During some malware outbreak period or intense malware activity, it has been observed on test platforms in our laboratory that most AV software are updated at least twice or three times a day.

[3] The term *malware pattern* or more precisely *malware detection pattern* will be preferred to the term *malware signature* or *signature* for short. Indeed, the latter term is used in other context like cryptography with a very specialized meaning.

Second generation detection techniques or heuristics aim at bypassing permutation σ while simple malware pattern scanning (first generation detection techniques) operates the detection itself in a final step.

Thus in our approach, we consider the indices in $\mathcal{F}$ where the bytes of $\mathcal{S}_\mathcal{M}$ are effectively located (up to a permutation) rather than the malware pattern bytes themselves. Our notation describes in a better way the action of any copycat which tries to produce an undetected malware variant without too much effort. In a context of black-box analysis, we denote $\mathcal{S}_{\mathcal{F},\mathcal{M}}$ the set of indices $\{i_1, i_2, \ldots, i_s\}$.

Let us now mathematically describe the action of a given malware detector $\mathcal{D}$. Let us first define the s binary variables X_j $(1 \leq j \leq s)$ as follows:

$$X_j = \begin{cases} 1 & \text{if } \mathcal{F}(i_j) = b_{\sigma(j)}, \\ 0 & \text{otherwise.} \end{cases}$$

This notation enables to clearly describe the modification of $\mathcal{S}_\mathcal{M}$ bytes by any copycat who tries to bypass a given detector $\mathcal{D}$. Thus X_j equals 0 means that the copycat has indeed modified $\mathcal{S}_\mathcal{M}(j)$.

Let us now consider a Boolean function $f_\mathcal{M} : \mathbb{F}_2^s \to \mathbb{F}_2$ where $\mathbb{F}_2$ is the binary finite field. We say that a given detector decides that $\mathcal{F}$ is infected by the malware $\mathcal{M}$ with respect to the *detection function* $f_\mathcal{M}$ and the malware pattern $\mathcal{S}_\mathcal{M}$ is and only if $f_\mathcal{M}(X_1, X_2, \ldots, X_s) = 1$. In other words:

$$f_\mathcal{M}(X_1, X_2, \ldots, X_s) = \begin{cases} 1 & \mathcal{F} \text{ is infected by } \mathcal{M}, \\ 0 & \mathcal{F} \text{ is not infected by } \mathcal{M}. \end{cases}$$

We will represent detection functions by their DNF, that is to say the logical disjunction of terms, each term being a conjunction of litterals X_i. By construction, literals do not appear under negated form $\overline{X_i}$. Thus, detection functions are modelled by *monotone* DNF. We will see in Sect. 3.1.2 that any scanning scheme can be described by a detection function $f_\mathcal{M}$.

Since copycats obviously show a great interest in the different ways of bypassing any given malware detection pattern, we will consider the *non-detection* function $\overline{f_\mathcal{M}}$ as the negation of the detection function $f_\mathcal{M}$ instead. In other words $\overline{f_\mathcal{M}} = 1 \oplus f_\mathcal{M}$. This function describes the way malware pattern bytes can be modified to bypass detection with respect to $f_\mathcal{M}$. These modifications correspond to the s-tuples $(x_1, x_2, \ldots, x_s)$ for which the non-detection equals 1. For a given such s-tuple, the modification that can be applied is defined as follows:

$$\begin{cases} \text{if } x_i = 0 & \text{byte } i \text{ in } \mathcal{S}_\mathcal{M} \text{ must be modified,} \\ \text{if } x_i = 1 & \text{byte } i \text{ in } \mathcal{S}_\mathcal{M} \text{ may be left unmodified.} \end{cases}$$

In the rest of the paper, we will indifferently consider either the detection function or the non-detection function.

Remark The malware $\mathcal{M}$ is uniquely characterized by both $\mathcal{S}_\mathcal{M}$ and the detection function $f_\mathcal{M}$. This function allows to greatly reduce the false positive rate.

In order to have a compact description of how sequence-based detection actually works, let us propose the two following definitions.

Definition 1 *A malware detection scheme with respect to a given malware $\mathcal{M}$ is the pair $\{\mathcal{S}_\mathcal{M}, f_\mathcal{M}\}$. If we consider sequence-based detection the set $\mathcal{S}_\mathcal{M}$ will contain bytes whereas if function-based detection is considered instead, this set contains program functions.*

Definition 2 *A malware bypassing scheme with respect to a given malware $\mathcal{M}$ is the pair $\{\mathcal{S}_\mathcal{M}, \overline{f_\mathcal{M}}\}$. If we consider sequence-based detection the set $\mathcal{S}_\mathcal{M}$ will contain bytes whereas if function-based detection is considered instead, this set contains program functions.*

2.2 Properties of malware patterns

Many questions arise about the kind of properties that a reliable malware detection pattern must exhibit. What are the best parameters to consider? Besides the problem of choosing good patterns, having some mathematical measures enables existing commercial scanners to be evaluated. As a matter of fact, antivirus software evaluation is mostly a subjective process which greatly differs from one test to another. Here are some properties that we have identified. Section 3 will be devoted to the accurate evaluation of these properties.

2.2.1 Malware pattern entropy and transinformation

The purpose is to determine the uncertainty that a detector black-box analyst must face when trying to extract a given malware pattern $\mathcal{S}_\mathcal{M}$ with respect to a given detector $\mathcal{D}$.

For that purpose, we use the entropy function as defined by Shannon [21]. Given a random variable X that takes a finite set of values with probabilities $p_1, p_2, \ldots, p_n$, the uncertainty or entropy of X is defined by:

$$H(X) = -\sum_{k=1}^{n} p_k \log_2(p_k).$$

In the present context, the variable X represents $\mathcal{S}_{\mathcal{F},\mathcal{M}}$ and any of its parameters. Indeed, the analyst has no information about any of the parameters of the detection pattern (its size s, the pattern bytes or equivalently their location in $\mathcal{F}$ and the detection function $f_\mathcal{M}$).

The difficulty to extract $\mathcal{S}_{\mathcal{F},\mathcal{M}}$ through a black-box analysis increases with $H(\mathcal{S}_{\mathcal{F},\mathcal{M}})$ and $H(\mathcal{S}_{\mathcal{F},\mathcal{M}}) = 0$ when no uncertainty exists.

Since in our case computing $H(X)$ is very complex, we will use the concept of *mutual information* or *transinformation* instead. If we consider that $\mathcal{S}_{\mathcal{M}}$ and $f_{\mathcal{M}}$ are random variables (for the analyst) we then define:

$$I\left(\mathcal{S}_{\mathcal{F},\mathcal{M}}; f_{\mathcal{M}}, \mathcal{S}_{\mathcal{M}}\right) = H\left(\mathcal{S}_{\mathcal{F},\mathcal{M}}\right) - H\left(\mathcal{S}_{\mathcal{F},\mathcal{M}} | f_{\mathcal{M}}, \mathcal{S}_{\mathcal{M}}\right)$$

as the amount of information that $f_{\mathcal{M}}$ and $\mathcal{S}_{\mathcal{M}}$ together reveal about $\mathcal{S}_{\mathcal{F},\mathcal{M}}$. In other words, the transinformation describes the consequence for the analyst to have a detector $\mathcal{D}$ at his disposal ($\mathcal{D}$ contains and leaks all the information about $f_{\mathcal{M}}$ and $\mathcal{S}_{\mathcal{M}}$).

By construction, it is obvious to define the same property when considering the non-detection function $\overline{f_{\mathcal{M}}}$. Then

$$I\left(\mathcal{S}_{\mathcal{F},\mathcal{M}}; f_{\mathcal{M}}, \mathcal{S}_{\mathcal{M}}\right) = I\left(\mathcal{S}_{\mathcal{F},\mathcal{M}}; \overline{f_{\mathcal{M}}}, \mathcal{S}_{\mathcal{M}}\right).$$

Malware pattern resistance

Once the analyst has succeeded in extracting the pattern bytes of a given detection scheme $\{\mathcal{S}_{\mathcal{M}}, f_{\mathcal{M}}\}$, he tries to bypass it in order to produce a non-detected variant. We then define the detection scheme *resistance*, denoted $\mathcal{R}(\mathcal{S}_{\mathcal{M}}, f_{\mathcal{M}})$ as the more or less difficulty that a copycat has to face in order to bypass it. This property obviously depends not only on the size of the set $\mathcal{S}_{\mathcal{M}}$, but also on the weight of the detection function $f_{\mathcal{M}}$ (denoted $\text{wt}\,(f_{\mathcal{M}}) = \left\{X \in \mathbb{F}_2^s | f_{\mathcal{M}}(X) = 1\right\}$):

- The number of possible modifications that enable the copycat to effectively produce a non-detected variant increase with the size of $\mathcal{S}_{\mathcal{M}}$;
- The lower the weight of the detection function, the easier it is to bypass the search mode with respect to $\mathcal{S}_{\mathcal{M}}$.

Thus, according to these characteristics, we define the scheme resistance as follows:

$$\mathcal{R}\left(\mathcal{S}_{\mathcal{M}}, f_{\mathcal{M}}\right) = \frac{\{X = (X_1, X_2, \ldots, X_s) | f_{\mathcal{M}}(X) = 0\}}{s2^s} \quad (1)$$

$$= \frac{\{X = (X_1, X_2, \ldots, X_s) | \overline{f_{\mathcal{M}}}(X) = 1\}}{s2^s}. \quad (2)$$

Consequently, we have:

$$0 \leq \mathcal{R}\left(\mathcal{S}_{\mathcal{M}}, f_{\mathcal{M}}\right) \leq 1,$$

The lower $\mathcal{R}\left(\mathcal{S}_{\mathcal{M}}, f_{\mathcal{M}}\right)$, the easier the detection scheme bypassing is. In the extreme, if $\mathcal{R}\left(\mathcal{S}_{\mathcal{M}}, f_{\mathcal{M}}\right) = 0$, there is no record for the malware $\mathcal{M}$ in the viral pattern database.

It it worth noticing that $\mathcal{R}(\mathcal{S}_{\mathcal{M}}, f_{\mathcal{M}})$ relates to the number of possible byte modification configurations (since $X_i \in \{0,1\}$). A copycat may modify every byte involved in a given configuration (e.g. an entry of the detection function) and replace it with any of the 255 remaining values. Thus, the effective grand total of possible byte modifications tremendously increases. This number conversely relates to $\mathcal{R}\left(\mathcal{S}_{\mathcal{M}}, f_{\mathcal{M}}\right)$. More precisely, if we denote $X = (X_1, X_2, \ldots, X_s) \in \mathbb{F}_2^s$ and if $\text{wt}(X)$ denote the Hamming weight of X (in other words the number of ones in the binary representation of X), then the grand total Δ of possible byte modifications that may be performed by a copycat is given by:

$$\Delta = \sum_{X \in \mathbb{F}_2^s} \overline{f_{\mathcal{M}}}(X).\left(256^{s-\text{wt}(X)} - 1\right). \quad (3)$$

We will examine some particular cases in Sect. 3.3.2.

2.2.2 Malware pattern efficiency

The efficiency of a malware pattern relates to its capacity to uniquely detect and identify a given malware. Moreover, any detection pattern must be frameproof with respect to other malware and any other (non-infected) file. This last requirement is linked both with false alarm probability (better known as false positive probability) and to malware phylogeny issues as defined in [10,12].

Previous results [11] showed that determining false positive probability is not as easy as it seems. Typically, malware pattern size ranges between 12 and 36 [11]. The probability for an s-byte sequence to be present in a text is $1/256^s$. Thus as soon as s is large enough, this probability tends towards 0 and theoretically no file could be falsely detected as infected unless it indeed contains the malware code. Unfortunately, theory greatly differs from practice. As pointed out in [11], there is a 34% chance of finding a random 24-byte sequence twice in a corpus of approximately half a gigabyte whereas theory of probability gives a 0.85×10^{-49}.

This discrepancy comes from the fact that bytes in an executable file cannot be described by independent, identically distributed variables (of probability $p = 1/256$. As an example, we have $P[b_0 =' M'] = 1$ and $P[b_1 =' Z' | b_0 =' M'] = 1$ in a Win32-executable. There exists a strong dependence between bytes in a executable code. Markov processes appear to be a better model when it comes to describing probabilistic behaviour of executable bytes.

As pointed out in [11], "*Cleverness, be it human or algorithmic, is an essential ingredient in choosing good*

computer viruses' signatures." Most commercial scanners use relatively efficient malware patterns, up to a more or less limited false positive probability.

Remark The malware pattern efficiency primarily depends on the pattern size. This is the reason why some commercial scanners are more sensitive to false positive probability than others due to pattern sizes which are too small. Moreover, the detection function may have some effect on the false positive probability as well. This point is still an open problem.

3 The detection pattern extraction problem and the mathematical evaluation of viral pattern scanners

3.1 Viral pattern extraction

AV scanners' efficiency has recently been studied by Christodorescu and Jha [2]. However their malware pattern extraction algorithm exhibit limitations. In addition to minor algorithmic flaws[4], their extraction algorithm considers only a limited number of detection function classes. We are now presenting two extraction algorithms, the second one being able to manage any detection function classes.

Given a malware detector $\mathcal{D}$, we aim at extracting the detection pattern of a given malware $\mathcal{M}$, with respect to $\mathcal{D}$. The extraction process is performed through a black-box analysis. No reverse-engineering is required. According to our previous notation, we must retrieve the non-detection function $\overline{f_{\mathcal{M}}}$ and the pattern bytes indices. Since the file $\mathcal{F}$ is the malware sample $\mathcal{M}$ itsef, we denote $\mathcal{S}_{\mathcal{M},\mathcal{M}}$ instead of $\mathcal{S}_{\mathcal{F},\mathcal{M}}$.

We consider the non-detection function rather than the detection function for the following reasons:

- In order to evaluate resistance of malware scanners with respect to black-box analysis, the copycat's point of view yields more information;
- Extracting the detection function would require to compute its negation in a second step. Unfortunately, computing the negation of a DNF formula has exponential worst case complexity.[5] Consequently, Algorithm E-2 will directly extract the non-detection function.

3.1.1 The naive approach: Algorithm E-1

A first algorithm has been considered in order to extract the most frequent detection pattern and the most common detection function. As compared with the algorithm given in [2], it may seem, at first sight, less efficient and rather naive. However, practical experience actually showed it was not. On the contrary, it succeeded in systematically extracting detection patterns as well as its corresponding detection function, whereas the Christodorescu and Jha's algorithm failed to do it in a few cases in which the detection pattern contained bytes scattered over the malware code.

Let us consider a malware code $\mathcal{M}$ of size n. Here follows our first extraction algorithm. The algorithm successively modifies each byte of the malware sample $\mathcal{M}$ (the byte is xored with a constant value) and performs a detection scanning with $\mathcal{D}$. If the detector still detects $\mathcal{M}$ (with our notation $\mathcal{D}(\mathcal{M}) = \sigma$), the modified byte is not involved in the detection pattern. On the contrary, the byte index belongs to $\mathcal{S}_{\mathcal{M},\mathcal{M}}$ (Table 1).

The complexity of Algorithm 1 is linear in the malware size, that is to say $\mathcal{O}(n)$. The main interest of Algorithm 1 lies in its capacity to manage malware pattern whatever their structure may be (sparse number of bytes, bytes randomly scattered over the code...). Unfortunately, its efficiency is limited to a single kind of detection function whose DNF formula is given by

$$f_{\mathcal{M}}(X_1, X_2, X_3, \ldots, X_s) = X_1 \wedge X_2 \wedge X_3 \wedge \cdots \wedge X_s.$$

This detection function corresponds to the most currently used detection technique: string scanning with no wildcards (first generation scanning techniques). We will use the AND function to denote this detection function. Contrary to what most commercial scanner publishers are generally claiming, the results show that this technique is still widely used. Appendix A presents detailed results about some variants of the *W32/Bagle* family.

Remark The non-detection function of the AND detection function is very easy to compute using simple Boolean calculus rules. We then have:

$$\overline{f_{\mathcal{M}}}(X_1, X_2, X_3, \ldots, X_s) = X_1 \vee X_2 \vee X_3 \vee \cdots \vee X_s.$$

This non-detection function is the OR function. Modifying any of the pattern bytes enables the malware to remain undetected.

[4] In particular, we identified some pattern configurations that lead to infinite loops in the FindLeftMost and FindRightMost procedures.

[5] Computing the negation of a DNF formula is equivalent to turn a conjuntive normal form into its corresponding disjunctive normal form. This transformation has exponential worst-case complexity [18, Theorem 4.1].

Table 1 Malware detection pattern extraction Algorithm E-1

Input: A malware sample $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$, a malware detector $\mathcal{D}$ and a malware name σ.
Output: The malware pattern $\mathcal{S}_{\mathcal{M},\mathcal{M}}$ (pattern indices).

$\mathcal{S}_{\mathcal{M},\mathcal{M}} \leftarrow \{\}$
for i = 1 to n do
  modify only byte m_i in $\mathcal{M}$
  If $\mathcal{D}(\mathcal{M}) \neq \sigma$ then
    $\mathcal{S}_{\mathcal{M},\mathcal{M}} \leftarrow \mathcal{S}_{\mathcal{M},\mathcal{M}} \cup \{m_i\}$
  end if
end for
return $\mathcal{S}_{\mathcal{M},\mathcal{M}}$

3.1.2 DNF formulae learning approach: Algorithm E-2

In order to extract any malware detection pattern and whatever the detection function may be (or equivalently the non-detection function), we are now considering another more powerful extraction algorithm based on learning techniques. Let us first recall some fundamental concepts in learning theory. The interested reader will refer to [13] for a detailed presentation on the subject.

We will focus on *Boolean concept learning* in which the learner's goal – in other words any black-box analysts in our context – is to infer how an unknown target function – the non-detection function in our case – classifies (as positive or negative) examples from a given domain. The *instance space* or *domain* $\mathcal{X}$ is the set of all possible objects (instances) to be classified – bytes of a malware detection pattern. We will consider the Boolean hypercube $\{0,1\}^n$ as reference domain. This corresponds to the non-detection function inputs as defined in Sect. 2.1. A *concept* is a Boolean function over domain $\mathcal{X}$. A *concept class* $\mathcal{C}$ is a collection of subsets of $\mathcal{X}$. In other words $\mathcal{X} \subseteq 2^{\mathcal{X}}$. In our case study, $\mathcal{C}$ describes the set of all possible non-detection function DNFs. Then each $x \in \mathcal{X}$ is classified according to membership of a *target concept* $f \in \mathcal{C}$ – the malware non-detection function $f_{\mathcal{M}}$. A point $x \in \mathcal{X}$ is a *positive* example of f if $f(x) = 1$ or a *negative* example of f otherwise. This case of learning method is denoted *Query Learning Model*.

In our context, the DNF formula to learn is the non-detection function $f_{\mathcal{M}}$ DNF. Each litteral X_i describes a byte of the malware sample $\mathcal{M}$ where $i = 1, 2, \ldots, n$ ($n = |\mathcal{M}|$). A DNF is then the union of conjunctions of litterals that may be either X_i or $\overline{X_i}$ (see in Appendix B the exact meaning of this notation).

It is worth noticing the following important point. The problem of learning general DNF formulae is a long-standing open problem even if efficient learning methods have been found for some particular DNF subclasses. However, the time and memory complexity of any learning algorithm obviously highly depends on the complexity of the underlying target concept. In the case of Boolean DNF formulae, it depends on the number of terms which ranges from 0 (the null function) to 2^n (the constant function $f(x) = 1$). The AND detection function that we have just considered in Sect. 3.1.1 contains a single term. This explains why Algorithm E-1 is suitable for this very particular function and is better than any other learning algorithm whose complexity is likely to be very high.

Our DNF formulae learning extraction algorithm is presented in Table 2. For the sake of clarity, we provide here a non-recursive version. The notation $X = 1_I(X_1, X_2, \ldots, X_n)$ describes the construction of a logical term in the DNF as follows using the characteristic function with respect to the interval I. Each litteral X_i or its negated form appears according to the following rule:

$$X_i = \begin{cases} \overline{X_i} & \text{if } i \in I, \\ X_i & \text{if } i \notin I. \end{cases}$$

Algorithm 2 is organised in three parts:

1. The first part is an initial learning step. It consists in modifying sections of contiguous bytes of malware code $\mathcal{M}$. These modifications are made through a dichotomic approach in $\log_2(n)$ steps. Sections of bytes are of decreasing length (a power of two). As in the Christodorescu and Jha's algorithm [2], the main purpose is to first identify the bytes which

Table 2 Malware detection pattern extraction Algorithm E-2

Input: A malware sample $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$, a malware detector $\mathcal{D}$ and a malware name σ.
Output: The malware pattern $\mathcal{S}_{\mathcal{M},\mathcal{M}}$ (pattern indices) and the non-detection function $\overline{f_{\mathcal{M}}}$ DNF.

$\mathcal{S}_{\mathcal{M},\mathcal{M}} \leftarrow \{\}$; $\mathrm{DNF}_{f_{\mathcal{M}}} \leftarrow \{\}$
$S \leftarrow 2^{\lfloor \log_2(n) \rfloor + 1}$; $S' \leftarrow 2^{\lfloor \log_2(n) \rfloor + 1}$
While $S > 0$ do
  $S \leftarrow \frac{S}{2}$; $q \leftarrow \frac{S'}{S}$
  For i = 0 to q -1 do
    binf $\leftarrow i \times S$; bsup $\leftarrow$ binf $+ S$
    modify bytes in $I =$ [binf, bsup[in $\mathcal{M}$
    If $\mathcal{D}(\mathcal{M}) \neq \sigma$ then
      $\mathcal{S}_{\mathcal{M},\mathcal{M}} \leftarrow \mathcal{S}_{\mathcal{M},\mathcal{M}} + I$
      $X = 1_I(X_1, X_2, \ldots, X_n)$
      $\mathrm{DNF}_{f_{\mathcal{M}}} \leftarrow \mathrm{DNF}_{f_{\mathcal{M}}} \cup X$
    End If
  End For
End While
$\mathcal{S}_{\mathcal{M},\mathcal{M}} \leftarrow$ CombinatorialMinimize($\mathcal{S}_{\mathcal{M},\mathcal{M}}$)
$\mathrm{DNF}_{f_{\mathcal{M}}} \leftarrow$ LogicalMinimize($\mathrm{DNF}_{f_{\mathcal{M}}}, \mathcal{S}_{\mathcal{M},\mathcal{M}}$)

are involved in the detection pattern $\mathcal{S}_{\mathcal{M},\mathcal{M}}$. This step enables some logical terms of the non-detection DNF to be found. This first step has complexity in $\mathcal{O}(2n)$ (While loop).

2. The second part, denoted CombinatorialMinimize, is a combinatorial minimisation step. Its aim is to eliminate the DNF terms which are redundant. The set $\mathcal{S}_{\mathcal{M},\mathcal{M}}$ which has been output by the previous step generally contains a few terms whose support interval (see previous notation) are included in some other support interval. To illustrate this point, logical terms $X_1X_2X_3X_4$ and X_1X_2 have support intervals $[1,4]$ and $[1,2]$, respectively. The second one is included in the first one. This implies that variables X_3 and X_4 are not involved in the detection pattern. We only keep the X_1X_2 term in the DNF. Thus, this combinatorial minimisation step is keeping the minimal elements in poset whose partial ordering is set inclusion. This step has a worst-case complexity in $\mathcal{O}(t\log(t) + tn)$ (a sort plus a comparison of consecutive terms) where $t = |\mathcal{S}_{\mathcal{M},\mathcal{M}}|$ before entering the CombinatorialMinimize procedure. The result is a set of size s.
3. The LogicalMinimize procedure whose purpose is twofold:
   - To complete the DNF learning process by exhaustively trying any s-tuple of variables. With the previous notations, to any s-tuple of $\mathbb{F}_2^s$ corresponds a byte modification configuration in $\mathcal{S}_{\mathcal{M},\mathcal{M}}$. The $\mathcal{M}$ code is then modified according this configuration and then tested with respect to the detector $\mathcal{D}$. If the code is no longer detected, the logical term corrsponding to this configuration is added to the DNF formula;
   - Then, a logical minimisation step is performed. Indeed, the output DNF contains terms which are logically redundant. Consequently, by applying Boolean calculus [5,17] and the Quine-McCluskey method to simplify Boolean expressions [15,19,20]. The logical minimisation problem is NP-hard [17, Sect. 5.8.3]. It has a complexity in $\mathcal{O}(s2^s)$ [26].

   Finally, the LogicalMinimize procedure has a complexity in $\mathcal{O}(2^s + s2^s)$ if $s = |\mathcal{S}_{\mathcal{M},\mathcal{M}}$ before entering this step.

As a conclusion, we have the following result.

Theorem 1 *The Algorithm 2 given in Fig. 2 succeeds in extracting the detection scheme of a malware $\mathcal{M}$ of size n whose detection pattern has size s with a time complexity in $\mathcal{O}(sn + s2^s)$.*

Proof When considering the previous elements, the general (worst-case) complexity is in $\mathcal{O}(2n + t\log(t) + tn + s^2 2^s)$. The terms $2n$ and t are negligible compared to 2^s and we have $t \equiv s$ as well (as confirmed by our experiments). Hence the result. □

Remarks

1. Algorithm 1 is a particular case of Algorithm 2 when the detection function is the AND function. Being

the most frequent case, using Algorithm 1 is a better choice since we suppress the logic and combinatorial minimization steps.
2. Learning the monotone DNF formula of the detection function requires a far better complexity when using Angluin's algorithm [1]. Using membership and equivalence queries, this algorithm exactly learns an unknown m-term monotone DNF formula over the domain $\{0,1\}^n$ with only $\mathcal{O}(nm)$ membership queries overall. But the DNF negation step must then be applied to get the non-detection function. This step has exponential worst-case complexity.
3. Algorithm E-2 succeeds in finding the exact non-detection function DNF unlike most learning methods which only produce equivalent function DNF.

3.2 Examples of detection functions

To any string based detection method corresponds a detection function DNF whose litterals are bytes of any malware undergoing a scanning process. In order to illustrate this point, let us consider some examples of pattern-based detection techniques that can be described by detection function DNFs which are different from the AND function.

3.2.1 Wildcard detection

Wildcard detection generally considers patterns in which variable bytes indices are involved. Let us consider the following detection pattern extract taken from [23, Sect. 11.1.2].

```
..... 07BB ??02 %3 33C9 ......
```

This detection pattern extract then is represented by the following detection function DNF (extract):

$$\begin{aligned}
&\ldots\ (X_i = 07) \wedge (X_{i+1} = BB) \wedge (X_{i+3} = 02)\\
&\qquad \wedge (X_{i+4} = 33) \wedge (X_{i+5} = C9)\\
&\vee\ (X_i = 07) \wedge (X_{i+1} = BB) \wedge (X_{i+3} = 02)\\
&\qquad \wedge (X_{i+5} = 33) \wedge (X_{i+6} = C9)\\
&\vee\ (X_i = 07) \wedge (X_{i+1} = BB) \wedge (X_{i+3} = 02)\\
&\qquad \wedge (X_{i+6} = 33) \wedge (X_{i+7} = C9)\\
&\ldots.
\end{aligned}$$

3.2.2 Techniques of mismatches

These techniques were developped by IBM as part of its antivirus scanner research. Techniques of mismatches allow N number of bytes in the detection pattern to represent any value, regardless of their byte location indices in the pattern. Let us consider the following detection pattern `01 02 03 04` with a mismatch value of 2 (this example has been extracted from [23, Sect. 11.1.3]. Then the corresponding detection function DNF (extract) is given by:

$$\begin{aligned}
&\ldots\ (X_i = 01) \wedge (X_{i+1} = 02) \vee (X_i = 01) \wedge (X_{i+2} = 03)\\
&\vee\ (X_{i+1} = 02) \wedge (X_{i+2} = 03) \vee (X_i = 01) \wedge (X_{i+3} = 04)\\
&\vee\ (X_{i+1} = 02) \wedge (X_{i+3} = 04) \vee (X_{i+2} = 03) \wedge (X_{i+3} = 04).
\end{aligned}$$

In this case, for a pattern of size s with mismatch value μ ($\mu < s$), it is obvious that the DNF formula contains $\binom{s}{\mu}$ terms, each of them having $s - \mu$ litterals (it is called a $(s-\mu)$-DNF).

3.2.3 Nearly exact identification

This method is used to increase the accuracy of detection. Instead of considering only one detection pattern, we consider two such patterns [23, Sect. 11.2.3]. In other words, we have to consider in this case two detection schemes $\left(\mathcal{S}^1_{\mathcal{M}}, f^1_{\mathcal{M}}\right)$ and $\left(\mathcal{S}^2_{\mathcal{M}}, f^2_{\mathcal{M}}\right)$. It is easy to prove that this case can be described by a unique scheme $(\mathcal{S}_{\mathcal{M}}, f_{\mathcal{M}})$ such that $\mathcal{S}_{\mathcal{M}} = \mathcal{S}^1_{\mathcal{M}} \cup \mathcal{S}^2_{\mathcal{M}}$ and $f_{\mathcal{M}} = f^1_{\mathcal{M}} \vee f^2_{\mathcal{M}}$ or $f_{\mathcal{M}} = f^1_{\mathcal{M}} \wedge f^2_{\mathcal{M}}$ according to the way both schemes are combined (up to a minimization step).

This case also applies when two or more detection engines are involved. Our model enables to consider a single engine instead.

3.2.4 Other detection techniques

Other sequence-based techniques like *bookmark techniques*, *smart scanning*, *skeleton detection*, *static decryptor techniques*, *sequence-based heuristics*..., and such like (see [23, Chap. 11]) can be similarly described. In these cases:

- The parameter s (the number of bytes involved) is very large (the detection pattern spans the whole code);
- The detection function may be generally far more complex (the number of logical terms is close to 2^s). However, it is not certain that the corresponding non-detection function is complex as well. This is an open problem (see Sect. 5); detection pattern spans the whole code);
- Particular combinatorial structures in the detection function DNF may be considered in addition to its weight.

As far as hashing or checksum techniques are considered, our approach can be easily generalized by considering bitwise approach instead of a bytewise approach. Any checksum value or hashing value can be described as a set of Boolean functions as demonstrated in [6]. The resulting detection function is then described as the logical intersection of these functions [8, Chap. 3].

3.3 Mathematical measures of AV scanners

Algorithms 1 and 2 enable to systematically extract detection pattern of the malware samples we used for testing (see Appendix A). The tested commercial scanners have shown some very interesting yet surprising results. As a general result, we can assert that the overall quality of detection pattern (including the detection and non-detection functions) tends to be weak, and, extremely weak in some cases. The main results are summarized hereafter.

3.3.1 Malware pattern transinformation

Whatever the commercial scanner we considered may be, the extraction of different malware pattern parameters (non-detection function $\overline{f_{\mathcal{M}}}$, pattern size s and pattern bytes indices set $\mathcal{S}_{\mathcal{M},\mathcal{M}}$) has been easily performed. In other words, as the scanner leaks all the information on the detection pattern, the transinformation value is consequently maximal. Thus we can write

$$I\left(\mathcal{S}_{\mathcal{M},\mathcal{M}};\overline{f_{\mathcal{M}}},\mathcal{S}_{\mathcal{M}}\right) = H(\mathcal{S}_{\mathcal{M},\mathcal{M}}).$$

This means that the only thing to do for a copycat who wants to get rid of the problem of uncertainty as far as a detection pattern is concerned, is to simply have the detector to bypass at one's disposal. In this respect, all the commercial scanners that we have tested are rather weak.

3.3.2 Malware pattern resistance

The scheme resistance quantifies the copycat's effort in bypassing a given detection scheme, once the latter has been successfully extracted. This property relates to the total number of possible byte modifications in order to delude a given detector $\mathcal{D}$. Equations 1 and 2, in Sect. 2.2.1 show that both the parameter s and the detection function $f_{\mathcal{M}}$ (or equivalently the non-detection function $\overline{f_{\mathcal{M}}}$) have an impact on this number.

The precise evaluation of the scheme resistance require to enumerate the set $\{X = (X_1, X_2, \ldots, X_s) | \overline{f_{\mathcal{M}}}(X) = 1\}$. Formula (3) in Sect. 2.2.1 then enables to compute the expected total number of possible byte modifications.

To summarize:

- As far as the AND detection function is concerned, when considering a detection pattern of size s, we have:
  $\Delta = 256^s - 1$.
  Consequently, short detection patterns are far better than longer ones. The latter yields more possible byte modifications, that may result in a non-detection;
- As far as other non-detection functions are concerned, evaluating Δ has a worst-case complexity in $\mathcal{O}(2^s)$. In this context, the copycat's effort increases with s. Long detection pattern are far better than short ones.
  The in-depth analysis of the antivirus that we have tested shows that detection pattern size are rather weak (about 15 bytes in average). Copycats' effort has consequently not to be very intense in practice, despite the overall complexity of Algorithm E-2. The parameter s should be larger ($s > 45$ at least).

Lastly, it is worth noticing that the weight of the non-detection function has a relatively high impact on the scheme resistance and on the parameter Δ. Since $\mathrm{wt}\left(\overline{f_{\mathcal{M}}}\right) + \mathrm{wt}\,(f_{\mathcal{M}}) = 2^s$, detection functions that we should consider in order to thwart copycat's action, are those that present a large weight. The experimental results that we have obtained with respect to the tested antivirus prove the latter to be rather weak when considering the resistance property.

The search and exploration of suitable detection functions are major issues and much research work must be carried out further. Another interesting point would be to determine whether some other structural properties can be considered in order to increase both detection efficiency and resistance against copycat's extraction and manipulations. Most of these issues still remain open problems.

3.3.3 Malware pattern efficiency

The black-box analysis revealed that the tested antivirus mostly used rather short malware detection patterns. As far as detection scheme efficiency is concerned, let us summarize our most important results and observations.

- Antivirus software can be divided into two groups, according to the average size s of the detection patterns:

  - Short-size patterns (from a few bytes to a few tens of bytes). This group contains most of the commercial products;
  - Long-size patterns (a few thousands of bytes). Only a few products belong to this group.

  We observed a trade-off between efficiency and resistance. As the resistance of the pattern grows, the number of false positives increase with it. This can be explained by the fact that detection function are AND functions. Other non-trivial functions should greatly enable to suppress this trade-off.
- The structure of the detection patterns (bytes involved, detection functions) presents striking similarities from one antivirus publisher to another (see Appendix A). This proves that malware analyses are obviously a joint work within the community of antivirus publishers. Moreover, contrary to what they generally claim, sharing information does not, as it is, contribute to make products any different – with the notable exception of *Avast* and *AVG*. The most relevant case is that of *F-Secure* and *Kaspersky* who share exactly the same detection patterns. The structural properties of a PE file cannot justify alone the very close similarities among these products.
- Whenever possible, heuristic engine have been tested separately. At least for the *W32/Bagle* family, heuristic detection does generally not provide any significantly better detection. The difference with other traditional engine proved to be rather disappointing.

4 Secure malware pattern scanning schemes

In this section, we present a new scanning scheme designed to detect malware that strongly limits copycats' analyses. By using suitable combinatorial structures to describe malware detection patterns as well as probabilistic approach to manage them, we can then thwart pattern black-box analysis and extraction under reasonable security assumptions. Unless a relatively high number of other analysts collude, it is impossible for any copycat to extract all the pattern parameters (pattern size, pattern bytes, non-detection function). Any of their potential attempts to produce malware variants from previous ones will be inevitably doomed to failure, thus contributing to strongly limited malware spreading.

4.1 Combinatorial and probabilistic constructions

4.1.1 General description

Our main purpose is to consider a general malware detection pattern of s bytes (generally scattered over the malware code). Whenever the scanning engine is involved in a detection process, only a sub-pattern of size k is used under the following constraints:

- The value k must be relatively small compared to s;
- Any sub-pattern is chosen at random;
- Any sub-pattern is a fixed value which depends on the user/computer identification data;
- The whole detection pattern cannot be reconstructed unless we know (extract) at least τ sub-patterns;
- The number π of sub-patterns and the value τ must be large.

These constraints have been chosen in such a way to maximize the copycat's uncertainty and effort when facing the malware detection pattern extraction problem. In other words, the copycat will not be able to reconstruct the whole pattern except under conditions that become very difficult to implement in practice. Lastly, if he succeeds in producing any variant which becomes undetected with respect to a given sub-pattern, the probability to remain detected for other sub-patterns (other users) must remain high. The proliferation of the variant will be, as a result, extremely limited.

The main issue was to find suitable combinatorial objects exhibiting the required properties. The best candidates are undoubtedly combinatorial designs [3,4]. Indeed, these objects show all the more interesting properties as their implementation only requires a limited amount of space/memory. The fourth constraint strongly suggested to consider (π, τ) threshold schemes [16, Chap. 12]. In our context, the secret to share is the whole detection pattern and each sub-pattern would then describe a share of the secret. Unfortunately, known threshold scheme contructions have been proposed when shares are numbers. In our case, the shares are more complex objects (typically collection of numbers). Some constructions have been proposed to extend classical threshold schemes to more complex objects like graphs [14]. However, up to now, these extensions require too much memory/space. Thus, they are not suitable for commercial antivirus products.

Let us notice that the third constraint aims at helping computer crime investigation in order to trace and identify copycats who would be responsible for the production and spreading of new variants.

4.1.2 Technical description of the scheme

In this respect, two aspects must be considered: the combinatorial object which describes the pattern $\mathcal{S}_{\mathcal{M}}$ itself and the detection function $f_{\mathcal{M}}$ (and consequently $\overline{f_{\mathcal{M}}}$).

The combinatorial object We have considered $2-(s,k,\lambda)$ designs (known as *Balanced Incomplete Block Designs* also or BIBD for short). For the sake of clarity, we will recall the definition of these objects as well as their most interesting properties.

Definition 3 *A balanced incomplete block design (BIBD) is a pair $(\mathcal{V},\mathcal{B})$ where $\mathcal{V}$ is a v-set and $\mathcal{B}$ is a collection of b k-subsets of $\mathcal{V}$ such that each element of $\mathcal{V}$ is exactly contained in r blocks and any 2-subset of $\mathcal{V}$ is contained in exactly λ blocks. The numbers v,b,r,k,λ are parameters of the BIBD.*

The following properties hold for a BIBD:

- *A BIBD exists if and only if $vr = bk$ and if $r(k-1) = \lambda(v-1)$.*
- *$r = (\lambda(v-1))/k - 1$.*
- *$b = vr/k$.*

An exhaustive presentation and classification of BIBDs as well as other combinatorial designs are provided in [3,4].

In our context, we have $\mathcal{S}_{\mathcal{M}} = \mathcal{V}, s = v, \pi = b = \lambda v(v-1)/k(k-1)$ and $\mathcal{B}$ describes the collection of sub-patterns $\mathcal{S}^i_{\mathcal{M}}$ with $\mathcal{S}_{\mathcal{M}} = \cup_i \mathcal{S}^i_{\mathcal{M}}$.

The detection function We choose a Boolean function $f : \mathbb{F}_2^s \to \mathbb{F}_2$ as the detection function. The DNF formula of $f_{\mathcal{M}}$ has weight 2^{s-1}. For any sub-pattern $\mathcal{S}^i_{\mathcal{M}}$, we then consider the restriction $f^i_{\mathcal{M}}$ of $f_{\mathcal{M}}$ with respect to $\mathcal{S}^i_{\mathcal{M}}$. The weight of $f_{\mathcal{M}}$ represents the maximal effort a copycat will face to extract the non-detection function (when considering the LogicalMinimize step) while maintaining a high level of pattern efficiency. According to the formalism developed in Sect. 2.1, the inputs of $f^i_{\mathcal{M}}$ are the k byte indices of the sub-pattern we consider. One of the best choices, from an algebraic and combinatorial point of view, is to consider the following function:

$$f(X_1, X_2, \ldots, X_s) = X_1 \oplus X_2 \oplus \cdots X_s.$$

This function can be implemented in a very reduced way (see Sect. 4.3). Let us notice that from the copycat's point of view, the only way to retrieve this (simplified) algebraic normal form is from the DNF. This transformation has a general complexity of $\mathcal{O}(2^s)$. It goes without saying that not only many other detection functions could be chosen but also π different detection functions may be considered, one for each sub-pattern.

The scanning protocol During its installation, the antivirus sofware gathered some information on both the system and the user:

- The serial number of the processor (CPUID) and that of the hard disk (HDID);
- The MAC address (denoted `MACadr`);
- The user name `USRname` and his email address `@adr`;
- A secret value hidden in the antivirus software ν.

Let us notice that other additional parameters can be considered.

Then, the sofware computes the sub-pattern index i as follows:

$$i = g(H(\text{CPUID} \oplus \text{HDID} \oplus \texttt{MACadr} \oplus \texttt{USRname} \oplus \texttt{@adr} \oplus \nu) \oplus \mathcal{N}).$$

The function H is a hash function whose input is the xor sum of the data encoded as integers while the function g outputs a random value which ranges from 1 to π, by means of a nonce $\mathcal{N}$. This value represents the indices of the detection sub-pattern that will be used for detection. The function g is chosen in such a way that the value is fixed for a given user from one scanning to another. The purpose is two-fold. Firstly, it ensures that a user (or a copycat) will always use the same sub-pattern. Lastly, in case of computer crime investigation, the investigator will be able to prove whether a suspected copycat is involved or not in the building of a new variant.

During each detection process in which scanning is involved, the software uses only the ith-sub-pattern along with the detection function f.

4.2 Mathematical analysis

Let us now analyze our scanning scheme. In order to make things clearer, let us note $\mathcal{S}_{\mathcal{M}}$, the whole detection pattern (in other words the $2-(s,k,\lambda)$ design we consider) whereas $\mathcal{S}^i_{\mathcal{M}}$ will denote its ith-sub-pattern and $f^i_{\mathcal{M}}$ the detection function taking the ith-sub-pattern as input.

Scheme transinformation According to our notation and the definition of the scheme, we obviously have:

$$I\left(\mathcal{S}_{\mathcal{M},\mathcal{M}}; \overline{f_{\mathcal{M}}}, \mathcal{S}_{\mathcal{M}}\right) = H\left(\mathcal{S}^i_{\mathcal{M},\mathcal{M}}\right) << H\left(\mathcal{S}_{\mathcal{M},\mathcal{M}}\right) - H\left(\mathcal{S}_{\mathcal{M},\mathcal{M}} | f^i_{\mathcal{M}}, \mathcal{S}^i_{\mathcal{M}}\right).$$

The black-box analyst manages to retrieve information on a very limited part of the detection pattern only.

Scheme resistance As previously stated, this property directly depends on both the weight of function wt $\left(f^i_{\mathcal{M}}\right)$ and k. For our scheme, we consequently have:

$$\mathcal{R}\left(\mathcal{S}^i_{\mathcal{M}}, f^i_{\mathcal{M}}\right) = \frac{2^k - 1}{k2^k} = \frac{1}{k} - \frac{1}{k2^k} = \mathcal{O}\left(\frac{1}{k}\right).$$

As a result, small values of k are better to get the best possible scheme resistance property (see Sect. 2.2.1).

Impact of copycat collusion We can imagine that a number of copycats try to collude in order to extract both

the whole detection pattern $\mathcal{S}_{\mathcal{M}}$ and detection function $f_{\mathcal{M}}$. Let us determine how many copycats must collude for that purpose.

Proposition 1 *$\mathcal{S}_{\mathcal{M}}$ and $f_{\mathcal{M}}$ can be extracted with a collusion of at least $\tau = \lceil s/k \rceil$ analysts.*

Proof The proof is obvious since each block contains k points and since that $\tau \geq \lceil s/k \rceil$. This corresponds to the situation in which the blocks involved in the collusion have a null intersection. In this case, the design is said resolvable.[6] □

When τ copycats collude, they then have to face a detection pattern of size s. As far as the detection function DNF formula is concerned, we obtain the following result:

Proposition 2 *The detection function DNF formula extract by a collusion of $\tau = \lceil s/k \rceil$ analysts contains at most $s/k\left(2^{k-1}\right)$ terms.*

Proof The proof is obvious since the intersection of any collection of blocks does not contain any block of the design and since any two blocks may have a non-empty intersection. There is equality only when blocks involved in the collusion are paiwise disjoint (parallel class). □

This implies that even τ analysts would collude, the complexity of the LogicalMinimize step would be intractable and thus the non-detection function could be retrieved.

New variant probability of non-detection Let us now suppose that a copycat succeeds in extracting both $\mathcal{S}^i_{\mathcal{M}}$ and $f^i_{\mathcal{M}}$ from a detector that implements our scanning scheme. He can thus produce a new variant from a known, detected one. What, then is, the probability for this variant to remain undetected while it is "*in the wild*"?

Proposition 3 *The knowledge of both $\mathcal{S}^i_{\mathcal{M}}$ and $f^i_{\mathcal{M}}$ enables to produce a variant which is still detected with a probability $P_{\text{detection}}$ such that*

$$\frac{1}{2} \leq P_{detection} \leq 1.$$

Proof Assume that the copycat uses $\mathcal{S}^i_{\mathcal{M}}$ and $f^i_{\mathcal{M}}$ to produce his variants. Let us now suppose that this variant has spread throughout a computer whose detector selects the jth sub-pattern. The modifications designed to produce the variant will affect the detection by means of the sub-pattern j with a probability equal to $1/2$ unless sub-patterns i and j have an empty intersection (detection probability is then equal to 1). Hence the result. □

This result shows that both the combinatorial object and the detection function must be carefully chosen. In particular, considering RBIBD is a more suitable choice than BIBD, even if in this case, experiments proved that $P_{\text{detection}}$ is closer to 1 than $1/2$. As a general result we have $P_{\text{detection}} = \pi - 1/\pi$ for RBIBD. Considering different detection functions, one for each sub-pattern is a better solution as well.

4.3 Implementation and performance results

The implementation of the proposed scheme is currently under investigation in our laboratory. We have considered different combinatorial objects. As far as security is concerned (especially the probability of non-detection for new variants), the best results have been obtained when considering parallel classes of RBIBD. Some interesting BIBD have also been satisfactorily used.

The upper bound of memory/space requirement is in $\mathcal{O}\left(s^2/k\right)$ when considering a general $2-(s,k,\lambda)$ design (essentially due to the size of the incidence matrix describing the design). Consideration about phylogeny aspects [10,12] should help to choose far better combinatorial designs which would enable several variants to be manage at once. As regards computing resources, we did not notice any significant increase in the scanning time.

Even though the implementation of our scheme needs to be developed further – so that it may become as efficient current scanners – our results turn out to be very promising insofar as this scheme can be undoubtedly considered as a practical solution.

5 Open problems, future work and conclusion

In the present paper, we have addressed both the problem about resistance against black-box analysis and classical detection scheme bypassing. We have first defined a new mathematical model for detection schemes that then enables us to define some properties that any reliable detection scheme should basically exhibit. Next, an in-depth analysis of a great deal of commonly used commercial antivirus products has been conducted with respect to this model and the relevant properties. The results of the experiments indicated that all the tested products turned out to be very weak. One of the obvi-

[6] In a *resolvable* BIBD, denoted RBIBD, the collection $\mathcal{B}$ can be partitioned into *parallel classes*. A *parallel class* is a set of blocks that partitions the point set $\mathcal{V}$. Two necessary conditions for BIBD to be resolvable are (1) $k|v$ and (2) $b \geq v + r - 1$.

Table 3 Tested antivirus software (versions and viral definitions)

Products	Version	Viral definition
Avast	4.6.691	0527–2
AVG	7.0.338	267.9.2/52
Bit Defender	7.2	09/16/05
DrWeb	4.33.2.12231	02/25/06 (104186)
eTrust	7.1.194	11.5.8400 (Vet)
eTrust	7.1.194	23.65.44 (InoculateIT)
F-Secure 2005	5.10–480	09/15/05
G-Data	AVK 16.0.3	KAV-6.818/BD-16.864
KAV Pro	5.0.383	09/19/05
McAfee 2006	–	DAT 4535
NOD 32	2.5	1.1189 (08/08/05)
Norton 2005	11.0.2.4	09/15/05
Panda Titanium 2006	5.01.02	01/17/06
Sophos	5.0.5 R2	3.98
Trend Office Scan	6.5 – 7.100	2.837.00

Table 4 Viral patterns for *W32/Bagle.A*

Product name	Signature size (in bytes)	Signature (indices)
Avast	29	14,128 → 14,144, 14,146 → 14,157, 14,159
AVG	7,297	0–1–60–200–201–220–469· · ·
Bit Defender	6	0–1–60–200–201–206
DrWeb	12	0–1–60–200–201–206–220· · · 222–461–465–469
eTrust/Vet	3,700	0–1–60–200–201–206–· · ·
eTrust/InoculateIT	3,700	0–1–60–200–201–206–· · ·
F-Secure 2005	11	0–1–60–200–201–206 220–240–241–461–469
G-Data	6	0–1–60–200–201–206
KAV Pro	11	The same as F-secure 2005
McAfee 2006	7	0–1–60–200–201–206–220
NOD 32	7,449	0–1–60–200–201–204–205· · ·
Norton 2005	0	(not an AND function)
Panda Tit. 2006	9	0–1–60–200–201–206 220–461–541
Sophos	28	0–1–60–200–201–206–220· · ·
Trend Office Scan	7	0–1–60–200–201–206–220

ous consequences of these investigations is that these existing weaknesses facilitate not only copycats' offences but also the spreading of variants derived from either original malware strains or previous variants.

We have presented a new scanning scheme that strongly prevents copycats' actions. Interestingly enough, not only it does succeed in highly limiting the spreading of variants, its implementation only requires a few computer resources (memory and computing time).

Lastly, we have identified a number of open problems that should motivate further research:

- The classification and exploration of good detection functions $f_{\mathcal{M}}$;
- Is it possible to find some structural properties of detection functions that would increase detection pattern efficiency while offering a high resistance level against black-box analysis. In particular, false positive probability could be significantly reduced thanks to detection functions which would be more appropriate than the AND function;
- Find other – more suitable – combinatorial objects [3,4] offering more interesting properties than those we used in our scanning scheme (pairwise designs, parallel designs...).

We have just begun applying our model and approach to the behaviour monitoring detection techniques. In this new context, the concept of sequence-based detection has been replaced with that of the function-based detection and instead of considering detection patterns,

Table 5 Viral patterns for *W32/Bagle.E*

Product name	Signature size (in bytes)	Signature (indices)
Avast	9	8,162–8,166–8,170–8,173–8,175 8,180–8,181–8,187–8189
AVG	13,288	0–1–60–216–217–222–236···
Bit Defender	87	2,831–2,964–···
DrWeb	1,773	0–1–60–216–217–222–236···
eTrust/Vet	4,306	0–1–60–216–217–222–···
eTrust/InoculateIT	4,306	0–1–60–216–217–222–···
F-Secure 2005	105	0–1–60–216–217–222–···
G-Data	3	2831–2964–2965
KAV Pro	105	The same as F-secure 2005
McAfee 2006	12,039	0–1–60–216–217–222–···
NOD 32	4,793	0–1–60–216–217–220–221···
Norton 2005	0	(not an AND function)
Panda Tit. 2006	37	0–1–59–60–216–217–222···
Sophos	15,028	0–1–2–4–8–12–13–16–24···
Trend Office Scan	146	0–1–60–216–217–222–···

Table 6 Viral patterns for *W32/Bagle.J*

Product name	Signature size (in bytes)	Signature (indices)
Avast	8	7,256 → 7,259 7,278 → 7,281
AVG	7,276	5739–5864–5866–···
Bit Defender	3,342	7,385–7,386–···
DrWeb	2,896	0–1–60–144–145–150–164···
eTrust/Vet	4,320	0–1–60–144–145–150–164···
eTrust/InoculateIT	1,311	0–1–60–144–145–150–164···
F-Secure 2005	3,128	0–1–60–144–145–150–164···
G-Data	2,954	0–1–60–144–145–150–445···
KAV Pro	3,128	The same as F-secure 2005
McAfee 2006	8,084	0–1–60–144–145–150–164···
NOD 32	9,629	0–60–144–145–148–149–···
Norton 2005	8	0–1–60–144–145 150–164–445
Panda Tit. 2006	437	0–1–32 → 35–60–64···
Sophos	3,429	0–1–60–144–145–150–164···
Trend Office Scan	63	7,750–···

we have, in the circumstances, considered malware behaviours. According to the very first investigations, there is every chance that our original model and approach may be fully transposable.

A Antivirus software analysis results

We provide here the results of the black-box signature extraction performed on some *Bagle* variants and for most existing commercial antivirus software (Table 3): All the variants of the *W32/Bagle* family have been used as viral samples for the signature extraction Algorithm E-1 which has been presented in Sect. 3.1.1. The naming convention for the viral variants is that used by VX Heavens [25], where we downloaded them in order to get a reference viral set (Tables 4, 5, 6, 7, 8).

Whenever several scanning engine were present, we have tested them separately when possible. The tables consider the best detection result.

Due to space limitations, the following tables give results for a few variants (exhaustive results are available upon request to the author). The table contains the indices of the bytes signature in the viral code but not the bytes themselves. As far as possible, the signature is given in full. If not, only a few beginning indices are provided in order to illustrate the "antivirus software phylogeny".

Table 7 Viral patterns for *W32/Bagle.N*

Product name	Signature size (in bytes)	Signature (indices)
Avast	10	14,567–14,574 → 14,577 14,581–14,585 → 14,588
AVG	13,112	533 → 663–665–···
Bit Defender	6,093	0–1–60–128–129–134–···
DrWeb	6,707	0–1–60–128–129–132–133···
eTrust/Vet	4,343	0–1–60–128–129–134–···
eTrust/InoculateIT	1,276	0–1–60–128–129–134–···
F-Secure 2005	54	0–1–60–128–129–548–···
G-Data	53	0–1–60–128–129–548–···
KAV Pro	54	The same as F-Secure 2005
McAfee 2006	4,076	0–1–60–128–129–134–···
NOD 32	17,777	0–1–60–128–129–132–133···
Norton 2005	51	0–1–60–128– 129–134–429–···
Panda Tit. 2006	1659	0–1–4–8–12–13–16–24···
Sophos	7,538	0–1–60–128–129–134–148···
Trend Office Scan	44	0–1–60–128–129–···

Table 8 Viral patterns for *W32/Bagle.P*

Product name	Signature size (in bytes)	Signature (indices)
Avast	8	12,916 → 12,919 12,937 → 12,940
AVG	14,575	533 → 536–538–···
Bit Defender	8,330	0–1–60–128–129–134–···
DrWeb	6,169	0–1–60–128–129–134–···
eTrust/Vet	1,284	0–1–60–128–129–134–···
eTrust/InoculateIT	1,284	0–1–60–128–129–134–···
F-Secure 2005	59	0–1–60–128–129–546–···
G-Data	54	0–1–60–128–129–546–···
KAV Pro	59	The same as F-Secure 2005
McAfee 2006	12,1278	0–1–60–128–129–134–···
NOD 32	21,849	0–1–60–128–129–132–133–···
Norton 2005	6	0–1–60–128–129–134
Panda Tit. 2006	7,579	0–1–60–134–148–182–209···
Sophos	8,436	0–1–60–128–129–134–148···
Trend Office Scan	88	0–1–60–128–129–···

B Norton Bagle.E non-detection function and pattern

Whereas the extraction Algorithm E1 did not produce any AND detection function for Norton on some *W32/Bagle* variants, Algorithm E2 systematically extracted quite easily the relevant data (detection pattern $\mathcal{S}_{\mathcal{M},\mathcal{M}}$ and the non-detection function $\overline{f_{\mathcal{M}}}$. In order to prevent any misuse, the minimized form of $\overline{f_{\mathcal{M}}}$ will not be given (that is to say, we do not give the result after the LogicalMinimize procedure action). However, it is worth noticing that from a logic minimization complexity point of view, Norton viral non-detection functions are rather weak. They depend only on a limited number of variables (or equivalently bytes; around between 15 and 25 according to the variants) and terms. Moreover, the function DNFs are very sparse. This weakness enables the copycat to easily bypass Norton's scanner.

The viral pattern extraction Algorithm E2 of Sect. 3.1 has produced the following non-detection function (before the logic minimization step) for *W32/Bagle.E* variant. The pattern depends on 15 bytes only, whose indices are

$$E = \{0, 1, 60, 61, 62, 63, 216, 217, 222, 223, 236, 237, 645, 646, 647, 648\}.$$

This result has been obtained once the COMBINATORIAL-MINIMIZE procedure action has occured. The notation x_i means that the byte of the pattern located at index i in the viral code, has not been modified whereas $\overline{x_i}$ means that it indeed has been.

Non-detection functions for other variants are available upon request to the author.

$$\begin{aligned}
&\overline{f_{\text{W32/Bagle.E}}} = \\
&\overline{x_0}.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648} \\
&\vee \overline{x_0}.\overline{x_1}.\overline{x_{60}}.\overline{x_{61}}.\overline{x_{62}}.\overline{x_{63}}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{646}.x_{646}.x_{647}.x_{648} \\
&\vee \overline{x_0}.\overline{x_1}.\overline{x_{60}}.\overline{x_{61}}.\overline{x_{62}}.\overline{x_{63}}.\overline{x_{216}}.\overline{x_{217}}.\overline{x_{222}}.\overline{x_{223}}.\overline{x_{236}}.\overline{x_{237}}.x_{645}.x_{646}.x_{647}.x_{648} \\
&\vee \overline{x_0}.\overline{x_1}.\overline{x_{60}}.\overline{x_{61}}.\overline{x_{62}}.\overline{x_{63}}.\overline{x_{216}}.\overline{x_{217}}.\overline{x_{222}}.\overline{x_{223}}.\overline{x_{236}}.\overline{x_{237}}.\overline{x_{645}}.\overline{x_{646}}.\overline{x_{647}}.\overline{x_{648}} \\
&\vee x_0.\overline{x_1}.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648} \\
&\vee x_0.x_1.\overline{x_{60}}.\overline{x_{61}}.\overline{x_{62}}.\overline{x_{63}}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648} \\
&\vee x_0.x_1.\overline{x_{60}}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648} \\
&\vee x_0.x_1.\overline{x_{60}}.\overline{x_{61}}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648} \\
&\vee x_0.x_1.\overline{x_{60}}.\overline{x_{61}}.\overline{x_{62}}.\overline{x_{63}}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648} \\
&\vee x_0.x_1.x_{60}.\overline{x_{61}}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648} \\
&\vee x_0.x_1.x_{60}.x_{61}.\overline{x_{62}}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648} \\
&\vee x_0.x_1.x_{60}.x_{61}.\overline{x_{62}}.\overline{x_{63}}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.\overline{x_{648}} \\
&\vee x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.\overline{x_{216}}.\overline{x_{217}}.\overline{x_{222}}.\overline{x_{223}}.\overline{x_{236}}.\overline{x_{237}}.x_{645}.x_{646}.x_{647}.x_{648} \\
&\vee x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.\overline{x_{216}}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648} \\
&\vee x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.\overline{x_{216}}.\overline{x_{217}}.\overline{x_{222}}.\overline{x_{223}}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648} \\
&\vee x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.\overline{x_{217}}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648} \\
&\vee x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.\overline{x_{222}}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648} \\
&\vee x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.\overline{x_{223}}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648} \\
&\vee x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.\overline{x_{236}}.x_{645}.x_{646}.x_{647}.x_{648} \\
&\vee x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.\overline{x_{237}}.x_{645}.x_{646}.x_{647}.x_{648} \\
&\vee x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{237}.\overline{x_{645}}.\overline{x_{646}}.\overline{x_{647}} \\
&\vee x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{237}.\overline{x_{645}}.\overline{x_{646}}.\overline{x_{647}}.x_{648} \\
&\vee x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{237}.\overline{x_{645}}.x_{646}.x_{647}.x_{648} \\
&\vee x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{237}.x_{645}.\overline{x_{646}}.x_{647}.x_{648} \\
&\vee x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{237}.x_{645}.\overline{x_{647}}.x_{648}
\end{aligned}$$

Acknowledgements Many thanks to Boris Sharov from *DrWeb* and to the *G-Data* company who provided me with an evaluation version of their antivirus.

References

1. Angluin, D.: Queries and concept learning. Mach. Learn. 2–4:319–342 (1988)
2. Christodorescu, M., Jha, S.: Testing malware detectors". In: Proceedings of the ACM International Symposium on Software Testing and Analysis (ISSTA'04) (2004)
3. Beth, T., Jungnickel, D., Lenz, H.: Design Theory, vol. 1. Cambridge: Cambridge University Press, ISBN 0-5214-4432-2 and ISBN 0-5217-7231-1 (1999)
4. Colbourn, C.J., Dinitz, J.H.: Handbook of Combinatorial Designs. Boca Raton: CRC Press, ISBN 0-8493-8948-8 (1996)
5. Denis-Papin, M., Kaufmann, A. et F. R.: Cours de calcul booléien appliqué, Coll. Bibliothèque de l'ingénieur électricien-mécanicien, Albin Michel éditeur (1963)
6. Filiol, E.: A new statistical testing for symmetric ciphers and hash functions. In: Proceedings of the 4th International Conference on Information and Communication Security 2002, Lecture Notes in Computer Science, vol. 2513, pp. 342–353. Berlin Heidelberg New York: Springer (2002)
7. Filiol, E.: Computer viruses: from theory to applications, IRIS international series. Berlin Heidelberg New York: Springer, ISBN 2-287-23939-1 (2005)
8. Filiol, E.: Advanced Computer Virology, IRIS International series. Berlin Heidelberg New York: Springer (to appear, 2006)
9. Fortinet, http://www.fortinet.com/FortiGuardCenter/global_threat_ stats.html
10. Goldberg, L.A., Goldberg, P.W., Phillips, C.A., Sorkin, G.B.: Constructing computer virus phylogenies. J. Algorithms **26**, 188–208 (1998)
11. Kephart, J.O., Arnold, W.: Automatic extraction of computer virus signatures. In: Proceedings of the 4th Virus Bulletin International Conference, pp. 179–194, Virus Bulletin Ltd (1994)
12. Karim, Md.E., Walenstein, A., Lakhotia, A.: Malware phylogeny generation using permutations of code. J. Comput. Virol. **1**(1–2), pp. 13–23 (2005)
13. Kearns, M., Vazirani, U.: An Introduction to Computational Learning Theory. MIT, Cambridge, ISBN 0-262-11193-4 (1994)
14. Kulesza, K., Kotulski, Z.: On secret sharing for graphs. arxiv.org/cs.CR/0310052 (2003)
15. McCluskey, E.J.: Minimization of Boolean functions. Bell Syst. Tech. J. **35**(5), 1417–1444 (1956)
16. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC, Boca Raton, ISBN 0-8493-8523-7 (1997)
17. Michaels, J.G.: Algebraic Structures. In: Rosen K.H. (ed.), Handbook of Discrete and Combinatorial Mathematics. pp. 344–354. CRC, Boca Raton, ISBN 0-8493-0149-1 (2000)
18. Papadimitriou, C.H.: Computational Complexity. Addison Wesley, Reading, ISBN 0-201-53082-1 (1995)
19. Quine, V.W.: The problem of symplifying truth functions. Am. Math. Monthly **59**(8), 521–531 (1952)
20. Quine, V.W.: A way to simplify truth functions. Am. Math. Monthly **62**, 627–631 (1955)
21. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**, pp. 379–423, 623–656 (1948)
22. Stimms, S., Potter, C., Beard, A.: 2004 information security breaches survey, UK Department of Trade and Industry, 2004. Available at http://www.security-survey.gov.uk. A video presenting the report to the press as well as a summary for decision-makers are also available on this website (2004)
23. Szor, P.: The Art of Computer Virus Research and Defense. Symantec Press and Addison Wesley, Reading, ISBN 9-780321-304544 (2005)
24. US Government Protection Profile: Anti-virus Applications for Workstations in Basic Robustness Environments, Version 1.0, January 2005. Available at www.iatf.net/protection_profiles/index.cfm
25. VX Heavens Database: vx.netlux.org
26. Wegener, I.: The complexity of Boolean functions. Wiley, New York (1987)